A 3D Desktop Puzzle Assembly System

Dmitri Shuralyov and Wolfgang Stuerzlinger (Team "York Red")

Dept. of Computer Science and Engineering, York University, Toronto, Canada

Abstract

We describe a desktop virtual reality system targeted at 3D puzzle assembly. The results of the evaluation show that all novices could successfully complete the puzzle within an average of about six minutes, while experts took about two minutes.

Keywords: 3D User Interfaces, 3D Manipulation

Index Terms: I.3.6 [Computer Graphics] Methodology and Techniques – Interaction Techniques, I.3.7 [Computer Graphics] 3D Graphics and Realism – Virtual Reality, H.5.2 [Human-Computer Interaction] User Interfaces.

1 System Description

Our main design choice was to use a desktop system, with a mouse as the input device. This was motivated by the fact that desktop systems and mice have (typically) lower latencies than other alternatives. Also, today there are good methods to map 2D input to 3D manipulation. In particular, we use a contract assumption, snapping, sliding, collision detection, interaction only with visible objects, and viewer constraints, as previous work [4] has shown all these to be beneficial. We display the 3D scene in perspective, but do not use stereo display, as it is to us (at best) unclear, if stereo has strong benefits for 3D interaction.

We used a PC with a 3.2 GHz QuadCore CPU, GeForce GTX470 graphics card, and 4 GB. For input we used a Logitech G9 Laser Precision Gaming Mouse and a 17" 1440x900 LCD for output.

We created an OpenGL C++ application, which re-implements the sliding algorithm of the SESAME system [3], but uses a new rotation and two new navigation methods. The user can toggle navigation methods via a key on the keyboard. In the experiment we used only the object-centric navigation.

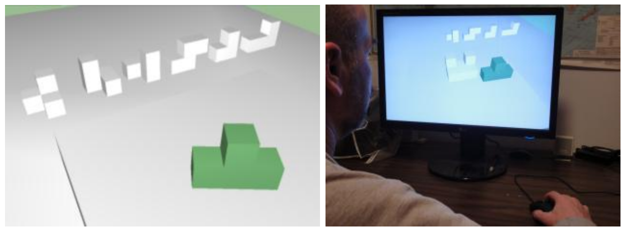
1.1.1 Translations

The main idea of the SESAME sliding algorithm [3] is that a manipulated object is always in contact with other (static) surfaces in the scene (unless over the background) and slides on these surfaces stably under the mouse cursor. Continuous collision detection with Opcode 1.3 prevents object interpenetration. With this, only 2D input is necessary to specify a 3D position, as the object can only move on the 2D manifold of visible surfaces.

This algorithm uses the frame buffer to detect all surface(s) *behind* the currently selected object. Then, the object is snapped in the view direction to the nearest surface behind it and slides on the corresponding background feature (face, edge, or vertex). Using the first surface *behind* the object ensures that this algorithm does not always "pop to front". This enables the user to slide an object under another. However, if the object disappears completely behind another object, i.e. no part of moving object is visible, the system "pops" the moving object to the front. Perspective correction ensures that the object remains stable under the cursor.

If a sliding object comes into contact with another object, the corresponding surface feature is added as a second movement constraint. E.g. if the object is in contact with both floor and wall, movements along the seam are directly mapped to cursor motion. Movements orthogonal to that, such as up the wall, are tolerated up to a screen-space threshold (50 pixels). Once that threshold is exceeded, the object is released from one constraint and e.g. starts to slide only on the wall. Multiple simultaneous contacts, such as a box in a corner, are handled similarly, except that a box in a corner snaps to a point constraint until the threshold is exceeded. Once an object slides "off" a contact surface the corresponding constraint is released and if the object ends up in mid air, the object snapped to the next surface behind it.

4700 Keele St., Toronto, ON, Canada, {shuryork,wolfgang}@cse.yorku.ca



Mouse button	Scene navigation	Object-centric
Left	Select & move	*same*
Right	Two-axis valuator rotate	*same*
Wheel	Rotate around view vector	*same*
Side button 1&2	Forward resp. backward	Orbit
Shift-Side button	Rotate view direction	Move in/out

Figure 1. Screenshot and photo of setup (currently selected object is green), mouse button assignments.

1.1.2 Rotations

Rotations are performed around the center of bounding box via quaternions. We map mouse motion to two axes of the view coordinate system, both orthogonal to the view vector to the center of the current object. During rotation we display both axes as guidance for the user. While this seems similar to an Arcball interface, this is a "two-axis valuator" interface, as defined by Bade *et al.* [1]. That paper also contains a user study whose results identify this as the best mouse-based rotation interface. In our implementation the third degree of freedom is mapped to the mouse wheel, which rotates the object around the view vector. To enable precise alignment and upon release of the mouse button, we snap the object to the nearest 30 degree multiple in *all three* Euler angles, in the global coordinate system. Rotation snapping is very important for lining up pieces precisely.

Moreover, we ensured that that the rotating object always remains in contact with the current surface below it, with collision avoidance, via the above-mentioned frame-buffer method with the view direction set to the contact normal. This results in the center of the object moving up and down relative to the contact plane.

1.1.3 Scene Navigation

The scene navigation mode is designed for moving around in a large environment. It enforces a constant viewer height above the current floor plane. For movement, we use the main method of the multi-scale 3D navigation approach [2]. This uses the depth buffer to identify nearby objects and computes an average movement vector based on the distances to all nearby objects. Essentially, holding down the "forward" button then moves the viewer towards the 3D point under the cursor, while avoiding collisions. Any mouse motion is mapped to view direction changes, which enables steering behaviour. The viewer stops at a safe distance before any surface. Holding down the "back" button reverses the computed direction, which enables the viewer to move back, again with collision avoidance. The means that moving back will move the viewer out the door and not through a wall. Changing the view direction is also possible by holding the Shift key simultaneously with a side button and moving the mouse. The view directions are limited to ±90 degrees up & down to prevent "flip-over". The viewer height can be changed via two keys.In object-centric navigation mode, pressing any side button and moving the mouse orbits around the center of the scene, limited to +90/-12 degrees. Shift-clicking the side button and moving the mouse forwards/backwards moves the viewer towards or away from the scene center.

2 User Study

Based on pilot tests, we decided fairly early on that we would provide assembly instructions. We did not intend to measure the puzzle solving time itself, as we wanted to focus on timing the 3D user interface, following the "spirit" of 3D user interface research. Consequently, we generated a printout of screenshots depicting the assembly sequence and put this beside the computer screen. This effectively removes the confound of cognitive puzzle-solving abilities, while still keeping the experiment susceptible to variability on 3D mental rotation tasks. Moreover, we were also interested in how our timings related to real-world puzzle assembly performance and we hence included a second task, namely the assembly of the "Google" puzzle (we did not have access to a physical version of the other puzzle). To make this condition a bit more comparable with the software, we again provide an assembly sequence, but allowed participants only to use two fingers of a single hand and limited manipulations to one piece at a time.

2.1 Study Details

We recruited ten novices and five 3D experts, mostly from the student population in the Department. We started each participant with the same configuration. Note that the initial position of the pieces matched neither the sequence nor the orientation of their final positions. Each participant was given a one-minute introduction to all facilities in the object-centric mode and allowed to practice for another minute. We also asked participants to work as fast and accurately as possible.

Then timing started with the first mouse/key action and ended at the end of the last mouse/key action before the user hit the Space key to signal that they were done. We choose to have the user decide when they were done as this avoids the issue of imperfect metrics for simultaneously matching translation and rotation. Also, note that the puzzle could be built at an arbitrary place *and rotation* on the plane, which makes automatic detection non-trivial. Instead, we computed the diagonal of the *oriented* bounding box around all pieces to determine how close the solution was to the optimum.

3 Results

The average participant age was 29.1, with a standard deviation (SD) of 11.2. Average computer use was 6.3 hours per day (SD 3.5) for novices and 11.8 (SD 3.4) for experts. Games were played 0.75 hours per week (SD 1.5) respectively 8 (SD 11.6). All participants were able to complete the puzzle.

The average time to completion for all novice users who completed the puzzle was 344 seconds (SD 147). Experts took 128 seconds on average (SD 28). A t-test reveals that these two are statistically different, with $p < 0.005$. The average oriented bounding box diagonal for the novices was 102.6% of the optimum (SD 2.2) and for experts 101.1% (SD 0.6). These two results are not statistically different. The timings for the real-world Google puzzle task show that novices took an average of 28.5 seconds (SD 10.6) and experts needed only 19.5 (SD 3.7). These two results are again statistically different, with $p < 0.05$.

3.1 Discussion

All users had no problems with translating objects, except when a puzzle piece had to be positioned between two other already placed (which was required once in our assembly sequence). With such a "tight" fit the collision detection library did not *reliably* permit the user to slide the piece into place. Hence, we verbally instructed the 2/3rd of participants who encountered this issue repeatedly to move one of the other pieces slightly and to slide the other piece in first. Also, we resorted to verbal instruction if users clearly had the wrong piece (about 1/3rd of all novices).

Most users made significant use of Object-Centric Navigation to get a better understanding of the object placement, and to get a better viewpoint for object manipulations.

The most important observation made during the experiment was that rotations take the majority of the time. We did not log event data, so we cannot quantify the percentage precisely, but we estimate that the average for rotations was more than two-thirds of the total, especially for novices. For a few novices, who got clearly stuck on rotations, we also resorted to a brief verbal instruction, such as "try moving up", to permit them to continue.

We recorded only a single attempt for each participant, even for the experts. Experts should achieve faster times with repetition and we hypothesize that they may even achieve one minute with training. But we do not expect that it is possible to match the speed of real-world puzzle assembly with this system.

Snapping to surfaces and rotations turned out to be instrumental and enabled all users to achieve quite high precision in the final puzzle configuration. Note that without snapping it is truly hard to place the pieces accurately. Snapping rotations at 30 degrees increments did not facilitate the task as much as one might think – with three angles, there are more snap points than initially apparent. In a pilot, we snapped to 90 degrees. This made the task noticeably simpler, but we consider this to be too task-specific.

In hindsight, we likely made the task of solving the puzzle harder than necessary for our participants, as we did not color the individual cubes, making it harder to find the right pieces.

References

[1] R. Bade, F. Ritter, B. Preim, Usability Comparison of Mouse-Based Interaction Techniques for Predictable 3D Rotation, Smart Graphics 2005, 138-150.

[2] J. McCrae, I. Mordatch, M. Glueck, A. Khan, Multiscale 3D navigation, Interactive 3D Graphics and Games 2009, 7-14.

[3] J.-Y. Oh, W. Stuerzlinger, J. Danahy, SESAME: Towards Better 3D Conceptual Design Systems, ACM DIS 2006, 80-89.

[4] W. Stuerzlinger, C. Wingrave, The Value of Constraints for 3D User Interfaces, Virtual Realities: Dagstuhl Seminar 2008, Springer Verlag, 2011, 203-224.